Laboratorio di Ingegneria del Software L-A

XML *Programming*

Argomenti

- XML in the .NET Framework
- XML Parsing Model
  - Tree-based parser – XMLDOM
  - Event-based parser – Simple API for XML (SAX)
- XML Reader
- XML Writer
- XML Document Object Model (DOM)

XML in the .NET Framework

- The XML classes in the .NET Framework provide a comprehensive and integrated set of classes, allowing you to work with XML documents and data
- XML classes in the .NET Framework can be broken into several groups
  - parsing and writing XML with the **`XmlReader`** and **`XmlWriter`**
  - validating XML with the **`XmlValidatingReader`**
  - editing an XML document using **`XmlDocument`**
  - performing XSL Transformations (XSLT) using **`XslTransform`**
  - editing XML Schema definition language (XSD) schema using **`XmlSchema`**
  - applying XPath queries using **`XPathNavigator`**

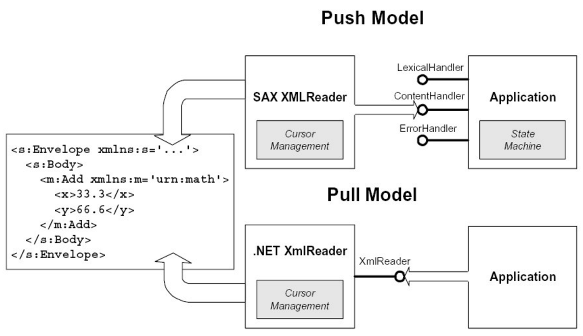
XML Parsing Model

- **Tree-based parser**
  - Parser **XMLDOM** – insieme di API che convertono un documento XML in una struttura (ad albero) in memoria
  - Per poter essere elaborato, il documento XML deve prima venire caricato completamente in memoria
- **Event-based parser**
  - Parser **SAX** – insieme di API in grado di elaborare gli elementi contenuti in uno stream di dati XML
    - Il parser controlla l'intero processo di scansione e invia (***PUSH***) i dati all'applicazione cliente (che decide cosa farne)
  - Per poter essere elaborato, il documento XML NON deve venire caricato completamente in memoria
  - Non direttamente disponibile in .NET

XML Parsing Model

- **XML Reader** (e **XML Writer**)
  - **Modello** di *parsing* di tipo ***PULL***
  - È possibile ottenere tutte le funzionalità del parser SAX
    - È sempre possibile costruire un modello *push* utilizzando un modello *pull*
  - Lavora sotto il totale controllo dell'applicazione cliente che può
    - ottenere solo i dati di interesse
    - saltando quelli privi di interesse
  - Le due classi (astratte) **`XmlReader`** e **`XmlWriter`** sono alla base di tutte le funzionalità XML in .NET (compreso XMLDOM)

Push Model vs Pull Model



```
<s:Envelope xmlns:s='...'>
  <s:Body>
    <m:Add xmlns:m='urn:math'>
      <x>33.3</x>
      <y>66.6</y>
    </m:Add>
  </s:Body>
</s:Envelope>
```

`System.IO` namespace

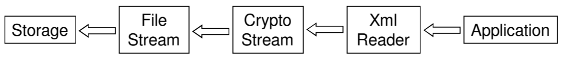
- The **`System.IO`** namespace contains types that allow **synchronous and asynchronous reading and writing** on
  - **data streams**
  - **files**
- A **file** is an ordered and named collection of a particular sequence of bytes having persistent storage – therefore, with files, one thinks in terms of **directory paths**, **disk storage**, and **file** and **directory names**
- A **stream** is an **abstraction of a sequence of bytes**, such as
  - a file
  - an input/output device
  - an inter-process communication pipe
  - a TCP/IP socket
  - …

`System.IO.Stream`

- The abstract base class **`Stream`** supports
  - **reading bytes** from a backing store
  - **writing bytes** to a backing store
- A **backing store** is a storage medium, such as a disk or memory
- Each different backing store implements its own stream as an implementation of the **`Stream`** class
- The **`Stream`** class and its derived classes provide a generic view of **data sources** and **repositories**, isolating the programmer from the specific details of the operating system and underlying devices

System.IO.Stream

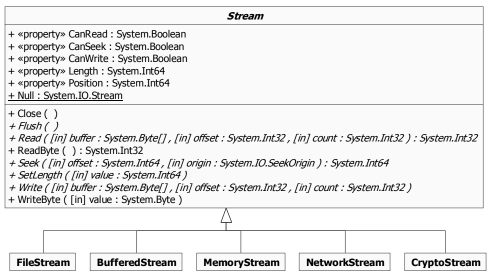
- Streams that connect to backing stores (**base streams**) have constructors that have the parameters necessary to connect the stream to the backing store
  - For example, **`FileStream`** has constructors that specify a path parameter, how the file will be shared by processes, and so on
- The design of the **`System.IO`** classes provides simplified **stream composing**
  - Base streams can be attached to one or more pass-through streams that provide the functionality you want
  - A **reader** or **writer** can be attached to the end of the chain so that the preferred types can be read or written easily



System.IO.Stream

- Streams involve these fundamental operations:
  - Streams can be read from – **reading** is the transfer of data from a stream into a data structure
  - Streams can be written to – **writing** is the transfer of data from a data source into a stream
  - Streams can support seeking – **seeking** is the querying and modifying of the current position within a stream
- Depending on the underlying data source or repository, streams might support only some of these capabilities (for example, NetworkStreams do not support seeking)
  - The **`CanRead`**, **`CanWrite`**, and **`CanSeek`** properties of **`Stream`** and its derived classes determine the operations that various streams support

System.IO.Stream

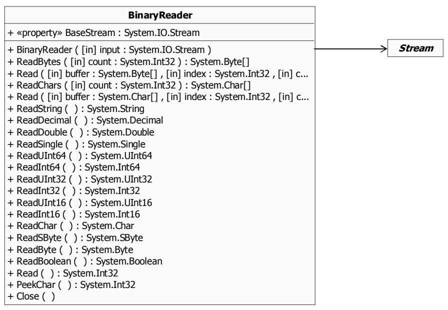

Stream
+ «property» CanRead : System.Boolean
+ «property» CanSeek : System.Boolean
+ «property» CanWrite : System.Boolean
+ «property» Length : System.Int64
+ «property» Position : System.Int64
+ Null : System.IO.Stream
+ Close ( )
+ Flush ( )
+ Read ([in] buffer : System.Byte[] , [in] offset : System.Int32 , [in] count : System.Int32) : System.Int32
+ ReadByte ( ) : System.Int32
+ Seek ([in] offset : System.Int64 , [in] origin : System.IO.SeekOrigin) : System.Int64
+ SetLength ([in] value : System.Int64)
+ Write ([in] buffer : System.Byte[] , [in] offset : System.Int32 , [in] count : System.Int32)
+ WriteByte ([in] value : System.Byte)
FileStream
BufferedStream
MemoryStream
NetworkStream
CryptoStream

System.IO.Stream

- **`FileStream`** – use this class to read from, write to, open, and close files on a file system, as well as to manipulate other file-related operating system handles such as pipes, standard input, and standard output
- **`MemoryStream`** – a stream whose backing store is memory
- **`NetworkStream`** – provides methods for sending and receiving data over sockets
- **`BufferedStream`** – adds a buffering layer to read and write operations on another stream
- **`CryptoStream`** – a stream that links data streams to cryptographic transformations
- **`Stream.Null`** – when the methods of **`Stream`** that provide writing are invoked on **`Null`**, the call simply returns, and no data is written; **`Null`** also implements a **`Read`** method that returns zero without reading data

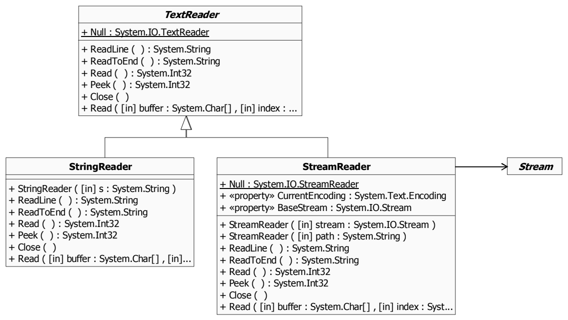
System.IO.BinaryReader System.IO.BinaryWriter

- **`BinaryReader`** and **`BinaryWriter`** read and write encoded strings and primitive data types from and to Streams


BinaryReader
+ «property» BaseStream : System.IO.Stream
+ BinaryReader ([in] input : System.IO.Stream)
+ ReadBytes ([in] count : System.Int32) : System.Byte[]
+ Read ([in] buffer : System.Byte[] , [in] index : System.Int32 , [in] c...
+ ReadChars ([in] count : System.Int32) : System.Char[]
+ Read ([in] buffer : System.Char[] , [in] index : System.Int32 , [in] c...
+ ReadString ( ) : System.String
+ ReadDecimal ( ) : System.Decimal
+ ReadDouble ( ) : System.Double
+ ReadSingle ( ) : System.Single
+ ReadUInt64 ( ) : System.UInt64
+ ReadInt64 ( ) : System.Int64
+ ReadUInt32 ( ) : System.UInt32
+ ReadInt32 ( ) : System.Int32
+ ReadUInt16 ( ) : System.UInt16
+ ReadInt16 ( ) : System.Int16
+ ReadChar ( ) : System.Char
+ ReadSByte ( ) : System.SByte
+ ReadByte ( ) : System.Byte
+ ReadBoolean ( ) : System.Boolean
+ Read ( ) : System.Int32
+ PeekChar ( ) : System.Int32
+ Close ( )
Stream

System.IO.TextReader System.IO.TextWriter

- **`TextReader`** and **`TextWriter`** read and write sequential series of characters from and to Streams


TextReader
+ Null : System.IO.TextReader
+ ReadLine ( ) : System.String
+ ReadToEnd ( ) : System.String
+ Read ( ) : System.Int32
+ Peek ( ) : System.Int32
+ Close ( )
+ Read ([in] buffer : System.Char[] , [in] index : ...
StringReader
+ StringReader ([in] s : System.String)
+ ReadLine ( ) : System.String
+ ReadToEnd ( ) : System.String
+ Read ( ) : System.Int32
+ Peek ( ) : System.Int32
+ Close ( )
+ Read ([in] buffer : System.Char[] , [in]...
StreamReader
+ Null : System.IO.StreamReader
+ «property» CurrentEncoding : System.Text.Encoding
+ «property» BaseStream : System.IO.Stream
+ StreamReader ([in] stream : System.IO.Stream)
+ StreamReader ([in] path : System.String)
+ ReadLine ( ) : System.String
+ ReadToEnd ( ) : System.String
+ Read ( ) : System.Int32
+ Peek ( ) : System.Int32
+ Close ( )
+ Read ([in] buffer : System.Char[] , [in] index : Syst...
Stream

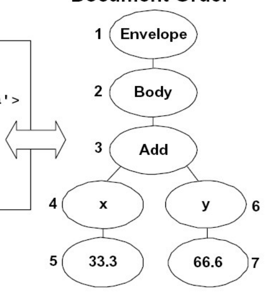
System.Xml.XmlReader System.Xml.XmlWriter

- XML-based I/O performed using a streaming interface suite
  - Models a **stream of logical XML nodes**
  - Streaming done in pull-mode (read) and push-mode (write)
- **`XmlReader`** models reading a stream of nodes
  - **`XmlReader`** is an abstract class
  - Fast, non-cached, forward-only, read-only access to XML data
  - Provides properties for inspecting current node
  - Nodes are processed in document order (depth-first traversal)
- **`XmlWriter`** models writing a stream of nodes
  - **`XmlWriter`** is an abstract class
  - Fast, non-cached, forward-only, write-only creation of XML data
  - Makes it easy to create well-formed XML data in a type-safe manner

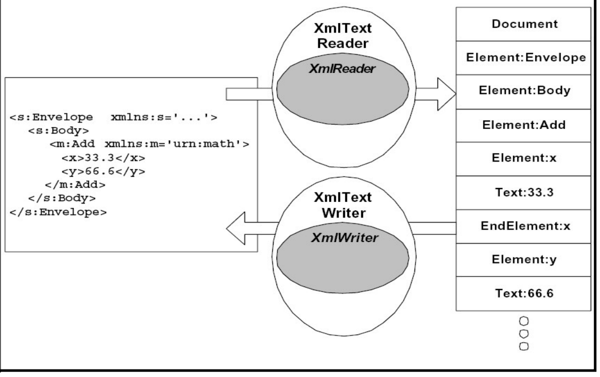
System.Xml.XmlReader System.Xml.XmlWriter

```
<s:Envelope xmlns:s='...'>
  <s:Body>
    <m:Add xmlns:m='urn:math'>
      <x>33.3</x>
      <y>66.6</y>
    </m:Add>
  </s:Body>
</s:Envelope>
```

Document Order



`System.Xml.XmlReader` `System.Xml.XmlWriter`


```
<s:Envelope  xmlns:s='...'>
  <s:Body>
     <m:Add xmlns:m='urn:math'>
       <x>33.3</x>
       <y>66.6</y>
     </m:Add>
  </s:Body>
</s:Envelope>
```

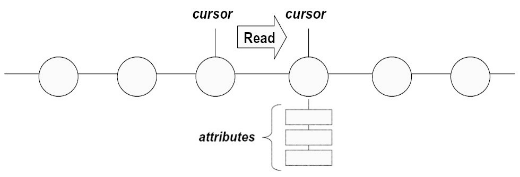
`System.Xml.XmlReader`

- Is an **abstract base** class that provides
  - **non-cached**
  - **forward-only**
  - **read-only**

  access to an XML stream
- Checks that the XML is **well-formed**, and throws **`XmlExceptions`** if an error is encountered
- Defines methods that enable you
  - to **pull data** from an XML source
  - to **skip unwanted nodes**

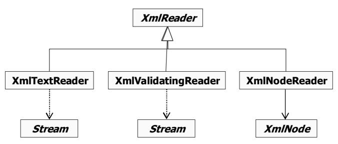
`System.Xml.XmlReader`

- Mantiene un puntatore interno (cursore) al nodo corrente (e ai suoi eventuali attributi)
- Non ha nozione del nodo precedente e del nodo successivo
- Non permette di modificare il nodo corrente
- Può solo far avanzare il cursore in avanti

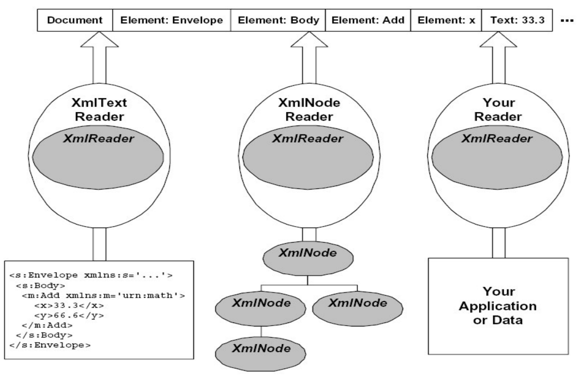


`System.Xml.XmlReader`

- Many implementations of **`XmlReader`** are possible
  - **`XmlTextReader`** uses a **`TextReader`** for I/O over XML 1.0
  - **`XmlValidatingReader`** provides DTD, XDR, and XSD validation while reading
  - **`XmlNodeReader`** uses an **`XmlNode`** as its input source
  - Custom readers can expose your own data as XML

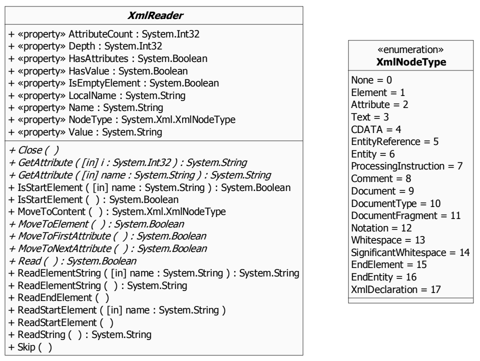


System.Xml.XmlReader



```
<s:Envelope xmlns:s='...'>
 <s:Body>
  <m:Add xmlns:m='urn:math'>
    <x>33.3</x>
    <y>66.6</y>
  </m:Add>
 </s:Body>
</s:Envelope>
```

System.Xml.XmlReader



XmlReader

```
+ «property» AttributeCount : System.Int32
+ «property» Depth : System.Int32
+ «property» HasAttributes : System.Boolean
+ «property» HasValue : System.Boolean
+ «property» IsEmptyElement : System.Boolean
+ «property» LocalName : System.String
+ «property» Name : System.String
+ «property» NodeType : System.Xml.XmlNodeType
+ «property» Value : System.String
+ Close ( )
+ GetAttribute ( [in] i : System.Int32 ) : System.String
+ GetAttribute ( [in] name : System.String ) : System.String
+ IsStartElement ( [in] name : System.String ) : System.Boolean
+ IsStartElement ( ) : System.Boolean
+ MoveToContent ( ) : System.Xml.XmlNodeType
+ MoveToElement ( ) : System.Boolean
+ MoveToFirstAttribute ( ) : System.Boolean
+ MoveToNextAttribute ( ) : System.Boolean
+ Read ( ) : System.Boolean
+ ReadElementString ( [in] name : System.String ) : System.String
+ ReadElementString ( ) : System.String
+ ReadEndElement ( )
+ ReadStartElement ( [in] name : System.String )
+ ReadStartElement ( )
+ ReadString ( ) : System.String
+ Skip ( )
```

«enumeration»
XmlNodeType

```
None = 0
Element = 1
Attribute = 2
Text = 3
CDATA = 4
EntityReference = 5
Entity = 6
ProcessingInstruction = 7
Comment = 8
Document = 9
DocumentType = 10
DocumentFragment = 11
Notation = 12
Whitespace = 13
SignificantWhitespace = 14
EndElement = 15
EndEntity = 16
XmlDeclaration = 17
```

Principali tipi di nodi XML

Tipo di nodo	Descrizione
Document	The container of all the nodes in the tree
XmlDeclaration	The declaration node: <?xml version="1.0"...>
Element	An element node: <item>
EndElement	An end element tag: </item>
Attribute	An attribute of an element: <… id="123">
Comment	A comment node: <!-- my comment -->
Text	Text belonging to an element or attribute
CDATA	A CDATA section <![CDATA[…my escaped text…]]>
Whitespace	An insignificant white space between markup text
SignificantWhitespace	An significant white space between markup text <item xml:space="preserve"> </item>

Lettura di un documento XML

1. **Creazione del reader** (scelte in alternativa):

```
XmlTextReader reader = new XmlTextReader(stream);
XmlTextReader reader = new XmlTextReader(textReader);
XmlTextReader reader = new XmlTextReader("nomeFile");
//  Per gestire in modo opportuno i whitespace
reader.WhitespaceHandling
  = WhitespaceHandling.All;  // default
  = WhitespaceHandling.Significant;
  = WhitespaceHandling.None;
```

2. **Scansione sequenziale dei nodi**:

```
while (reader.Read())
{
  … accesso al nodo corrente …
}
```

3. **Chiusura del reader**:

```
reader.Close();
```

Lettura dei nodi

`WhitespaceHandling.All`

```
  <author>Carson</author>
ΔΔ<author>ΔΔΔ</author>
ΔΔ<author xml:space="preserve">ΔΔΔ</author>
                    ⇩
NodeType=Element, name="author", value=""
NodeType=Text, name="", value="Carson"
NodeType=EndElement, name="author", value=""
NodeType=Whitespace, name="", value="
ΔΔ"
NodeType=Element, name="author", value=""
NodeType=Whitespace, name="", value="ΔΔΔ"
NodeType=EndElement, name="author", value=""
NodeType=Whitespace, name="", value="
ΔΔ"
NodeType=Element, name="author", value=""
NodeType=SignificantWhitespace, name="", value="ΔΔΔ"
NodeType=EndElement, name="author", value=""
```

Laboratorio di Ingegneria del Software L-A

Lettura dei nodi

`WhitespaceHandling.Significant`

```
  <author>Carson</author>
ΔΔ<author>ΔΔΔ</author>
ΔΔ<author xml:space="preserve">ΔΔΔ</author>
                    ⇩
NodeType=Element, name="author", value=""
NodeType=Text, name="", value="Carson"
NodeType=EndElement, name="author", value=""
NodeType=Element, name="author", value=""
NodeType=EndElement, name="author", value=""
NodeType=Element, name="author", value=""
NodeType=SignificantWhitespace, name="", value="ΔΔΔ"
NodeType=EndElement, name="author", value=""
```

Laboratorio di Ingegneria del Software L-A

Lettura dei nodi

`WhitespaceHandling.None`

```
  <author>Carson</author>
ΔΔ<author>ΔΔΔ</author>
ΔΔ<author xml:space="preserve">ΔΔΔ</author>
                    ⇩
NodeType=Element, name="author", value=""
NodeType=Text, name="", value="Carson"
NodeType=EndElement, name="author", value=""
NodeType=Element, name="author", value=""
NodeType=EndElement, name="author", value=""
NodeType=Element, name="author", value=""
NodeType=EndElement, name="author", value=""
```

Laboratorio di Ingegneria del Software L-A

Lettura dei nodi

`WhitespaceHandling.None`

```
  <author>&#32;&#32;&#32;</author>
                    ⇩
NodeType=Element, name="author", value=""
NodeType=Text, name="", value="ΔΔΔ"
NodeType=EndElement, name="author", value=""


  <author>&nbsp;&nbsp;&nbsp;</author>
                    ⇩
NodeType=Element, name="author", value=""
NodeType=EntityReference, name="nbsp", value=""
NodeType=EntityReference, name="nbsp", value=""
NodeType=EntityReference, name="nbsp", value=""
NodeType=EndElement, name="author", value=""
```

Laboratorio di Ingegneria del Software L-A

Esempio di lettura di nodi

```
while (reader.Read())
{
  switch (reader.NodeType)
  {
    case XmlNodeType.Element:
      … elaborazione apertura nodo di tipo element
      break;
    case XmlNodeType.EndElement:
      //  Solo con </Element>, non nel caso <Element />
      … elaborazione chiusura nodo di tipo element
      break;
    default:
      //  Probabilmente, gli altri tipi di nodo non interessano
      break;
  }
}
```

Esempio di lettura di nodi

```
<?xml version="1.0" encoding="utf-8" ?>
<!- Commento -->
<Gruppo>
   <Item nome="Pippo" />
   <Item nome="Topolino"></Item>
   <Item nome="Paperino" />
   <Item nome="Gastone" />
</Gruppo>

XmlTextReader reader = new XmlTextReader(...);
reader.WhitespaceHandling = WhitespaceHandling.None;
reader.MoveToContent();  //  Salta commenti e dichiarazioni
reader.ReadStartElement("Gruppo");
while (reader.IsStartElement("Item"))
{
   ...  //  Elabora Item
   reader.Skip();  //  Cosa succede con Read()?
}
reader.ReadEndElement();
reader.Close();
```

Metodi utili

- `XmlNodeType MoveToContent()`
  Salta commenti e dichiarazioni
- `void ReadStartElement()`
  `void ReadStartElement(string name)`
  Se il nodo corrente è l'apertura di un elemento (di nome "`name`"), il *reader* si posiziona sul nodo successivo, in caso contrario, `XmlException`
- `void ReadEndElement()`
  Se il nodo corrente è la chiusura di un elemento, il *reader* si posiziona sul nodo successivo, in caso contrario, `XmlException`
- `void Skip()`
  Salta sia tutti i figli del nodo corrente, sia l'eventuale chiusura

Esempio di lettura di nodi

```
<?xml version="1.0" encoding="utf-8" ?>
<!- Commento -->
<Gruppo>
   <Item>Pippo</Item>
   <Item>Topolino</Item>
   <Item>Paperino</Item>
   <Item>Gastone</Item>
</Gruppo>

XmlTextReader reader = new XmlTextReader(...);
reader.WhitespaceHandling = WhitespaceHandling.None;
reader.MoveToContent();  //  Salta commenti e dichiarazioni
reader.ReadStartElement("Gruppo");
while (reader.IsStartElement("Item"))
{
   ... = reader.ReadString(); // Elabora contenuto di Item
   reader.Skip();
}
reader.ReadEndElement();
reader.Close();
```

Metodi utili

- **`string ReadString()`**
  Restituisce il contenuto testuale di un nodo di tipo "Element" o "Text" – non modifica la posizione del reader, ma consuma l'informazione!
- **`string ReadElementString()`**
  `string ReadElementString(string name)`
  Restituisce il contenuto testuale di un semplice elemento con solo testo – salta anche la chiusura dell'elemento

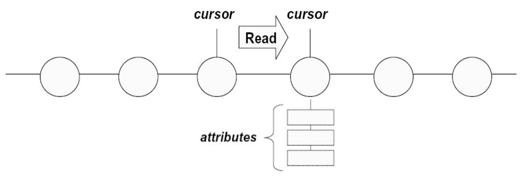
```
reader.ReadStartElement("Gruppo");
while (reader.IsStartElement("Item"))
{
   ... = reader.ReadElementString();
}
reader.ReadEndElement();
```

Laboratorio di Ingegneria del Software L-A

4.33

Lettura degli attributi

- Solo i nodi di tipo **`Element`**, **`DocumentType`** and **`XmlDeclaration`** possono avere attributi
- Gli attributi NON fanno parte dello stream principale di nodi XML
- **`bool HasAttributes`**
  restituisce **`True`** se il nodo corrente ha almeno un attributo
- **`int AttributeCount`**
  restituisce il **numero di attributi** del nodo corrente



Laboratorio di Ingegneria del Software L-A

4.34

Lettura degli attributi

- **`string GetAttribute(int index)`**
  `string GetAttribute(string name)`
  restituiscono il **valore di un attributo**, dato l'indice o il nome

```
if(reader.HasAttributes)
{
  for (int k = 0; k < reader.AttributeCount; k++)
  {
    //  Non è possibile ottenere il nome dell'attributo!
    Console.WriteLine("Attribute value=\"{0}\"",
      reader.GetAttribute(k));
  }
}

… = reader.GetAttribute("NomeAttributo");  //  Sì
```

- Da utilizzare per ottenere il valore di un attributo, conoscendone il nome
- Se non esiste un attributo con il nome passato come argomento, viene restituito `null`

Laboratorio di Ingegneria del Software L-A

4.35

Lettura degli attributi

- **`bool MoveToNextAttribute();`**
  permette di scandire con il reader tutti gli attributi del nodo corrente
- **`bool MoveToElement();`**
  riposiziona il reader sul nodo di partenza (cioè, quello che contiene la lista degli attributi)

```
if(reader.HasAttributes)
{
  while (reader.MoveToNextAttribute())
  {
    Console.WriteLine("Attribute name=\"{0}\",
      value=\"{1}\"", reader.LocalName, reader.Value);
  }
  reader.MoveToElement();
}
```

- Da utilizzare per scandire l'intera lista di attributi

Esempio 2

Laboratorio di Ingegneria del Software L-A

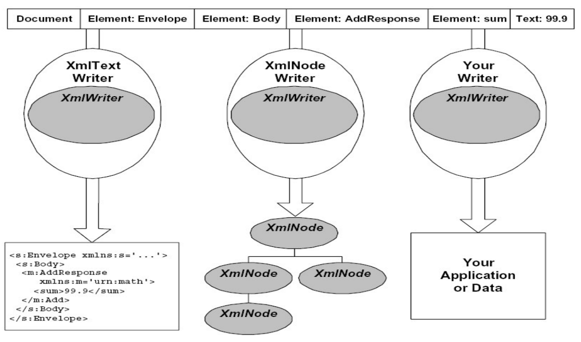
4.36

System.Xml.XmlWriter

- Is an **abstract base** class that provides a
  - **fast**
  - **non-cached**
  - **forward-only**

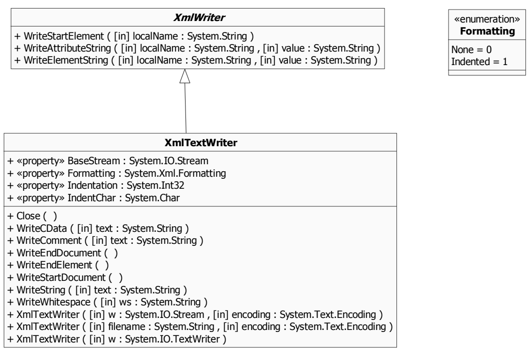
  means of generating streams containing XML data that conforms to
  - the W3C Extensible Markup Language (XML) 1.0 and
  - the Namespaces in XML recommendations
- Many implementations of `XmlWriter` are possible
  - `XmlTextWriter` uses a `TextWriter` for I/O
  - Custom writers

System.Xml.XmlWriter



```
<s:Envelope xmlns:s='...'>
 <s:Body>
  <m:AddResponse
     xmlns:m='urn:math'>
    <sum>99.9</sum>
  </m:Add>
 </s:Body>
</s:Envelope>
```

System.Xml.XmlWriter



```
XmlWriter
+ WriteStartElement ( [in] localName : System.String )
+ WriteAttributeString ( [in] localName : System.String , [in] value : System.String )
+ WriteElementString ( [in] localName : System.String , [in] value : System.String )

«enumeration»
Formatting
None = 0
Indented = 1

XmlTextWriter
+ «property» BaseStream : System.IO.Stream
+ «property» Formatting : System.Xml.Formatting
+ «property» Indentation : System.Int32
+ «property» IndentChar : System.Char
+ Close (  )
+ WriteCData ( [in] text : System.String )
+ WriteComment ( [in] text : System.String )
+ WriteEndDocument (  )
+ WriteEndElement (  )
+ WriteStartDocument (  )
+ WriteString ( [in] text : System.String )
+ WriteWhitespace ( [in] ws : System.String )
+ XmlTextWriter ( [in] w : System.IO.Stream , [in] encoding : System.Text.Encoding )
+ XmlTextWriter ( [in] filename : System.String , [in] encoding : System.Text.Encoding )
+ XmlTextWriter ( [in] w : System.IO.TextWriter )
```

Scrittura di un documento XML

1. **Creazione del writer** (scelte in alternativa):
   ```
   XmlTextWriter writer =
     new XmlTextWriter(textWriter);
     new XmlTextWriter(stream, encoding);
     new XmlTextWriter("nomeFile", encoding);
   writer.Formatting = Formatting.Indented;
   writer.Indentation = 2;  //  default
   ```
2. **Scrittura del documento**:
   ```
   // produce: <?xml version="1.0"?>
   writer.WriteStartDocument();
   while (...)
   {
     … scrittura nodo …
   }
   writer.WriteEndDocument();
   ```
3. **Chiusura del writer**:
   ```
   writer.Close();
   ```

Scrittura di un elemento

- `<Item>…</Item>`

```
writer.WriteStartElement("Item");
//  scrittura eventuali attributi
…
writer.WriteEndElement();
```

- `<Item>Testo</Item>`

```
writer.WriteStartElement("Item");
//  scrittura eventuali attributi
writer.WriteString("Testo");
writer.WriteEndElement();

//  se non esistono attributi
writer.WriteElementString("Item","Testo");
```

Scrittura di un attributo

- `<Item nome="valore">…</Item>`

```
writer.WriteStartElement("Item");
writer.WriteAttributeString("nome","valore");
…
writer.WriteEndElement();
```

- `<Item nome="valore"/>`

```
writer.WriteStartElement("Item");
writer.WriteAttributeString("nome","valore");
writer.WriteEndElement();
```

Esempio 2

`System.Xml.XmlConvert`

- Provides methods that enable you to convert from a string to a .NET Framework data type and vice-versa
- **Locale settings are not taken into account during data conversion**
- The data types are based on the XML Schema (XSD) data types

Tipo di dato	`XmlConvert`	`Convert` `ToString` e `Parse`
`bool`	`true` / `false` legge anche: `1` / `0`	`True / False`
`float` e `double`	`3.14`	`3,14`
`DateTime`	`2004-05-09 T00:00:00.0000000+ 02:00`	`09/05/2004 0.00.00`

XML Document Object Model

- An in-memory representation of an XML document
- The DOM allows you to programmatically
  - Load
  - Modify
  - Save

  an XML document

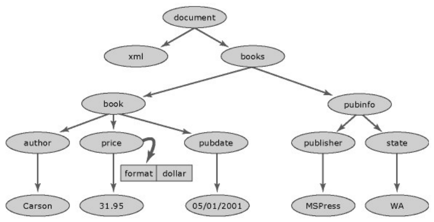
- The `XmlReader` class also reads XML, however
  - it provides non-cached, forward-only, read-only access

  this means that
  - there are no capabilities to edit the values of an attribute or content of an element, or the ability to insert and remove nodes

Laboratorio di Ingegneria del Software L-A

XML Document Object Model

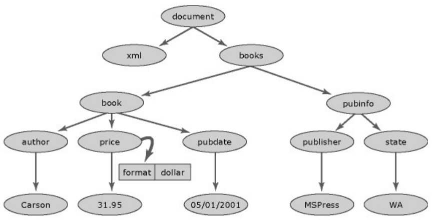
```
<?xml version="1.0"?>
<books>
  <book>
    <author>Carson</author>
    <price format="dollar">31.95</price>
    <pubdate>05/01/2001</pubdate>
  </book>
  <pubinfo>
    <publisher>MSPress</publisher>
    <state>WA</state>
  </pubinfo>
</books>
```



4.45

Laboratorio di Ingegneria del Software L-A

XML Document Object Model

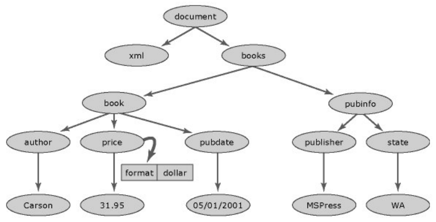


- Nodes have a single **parent node**, a parent node being a node directly above it (the only node that do not have a parent is the “document” node)
- Most nodes can have multiple **child nodes**, which are nodes directly below it
- Nodes that are at the same level are **siblings** (the “book” and “pubinfo” nodes, …)

4.46

Laboratorio di Ingegneria del Software L-A

XML Document Object Model



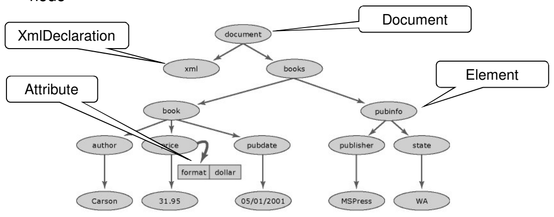
- Gli attributi non fanno parte delle relazioni *parent*, *child* e *sibling*
- Gli attributi vengono considerati **proprietà dei nodi di tipo element**, e sono costituiti da una **coppia nome-valore**
- Nell’esempio:
  - la parola “**`format`**” è il nome dell’attributo
  - la stringa “**`dollar`**” è il valore dell’attributo `format`

4.47

Laboratorio di Ingegneria del Software L-A

XML Document Object Model

- As XML is read into memory, nodes are created
- However, not all nodes are the same type
- An element, in XML, has different rules and syntax than a processing instruction
- So as various data is read, a **node type** is assigned to each node
- This node type determines the characteristics and functionality of the node



4.48

XML Document Object Model

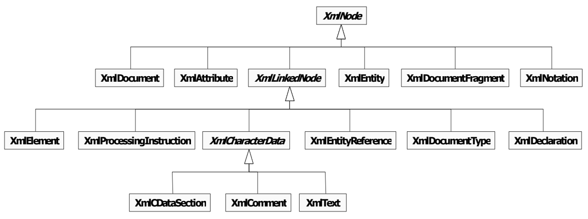
DOM Node Type	Classe	Descrizione
Document	`XmlDocument`	The container of all the nodes in the tree
Element	`XmlElement`	Represents an element node
Attr	`XmlAttribute`	Is an attribute of an element
Comment	`XmlComment`	A comment node
Text	`XmlText`	Text belonging to an element or attribute
CDATASection	`XmlCDataSection`	Represents CDATA
Declaration	`XmlDeclaration`	Represents the declaration node <?xml version="1.0"...>

XML Document Object Model

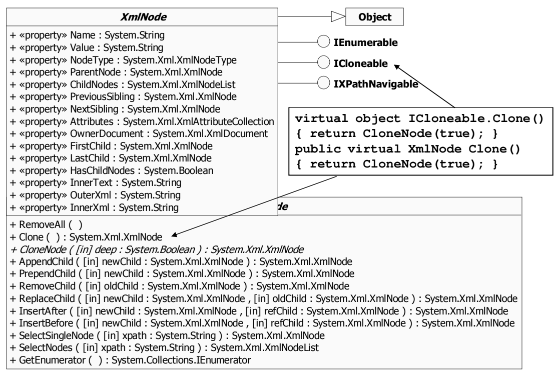
DOM Node Type	Classe	Descrizione
DocumentFragment	`XmlDocumentFragment`	A temporary bag containing one or more nodes without any tree structure
DocumentType	`XmlDocumentType`	Represents the <!DOCTYPE…> node
EntityReference	`XmlEntityReference`	Represents the non-expanded entity reference text
ProcessingInstruction	`XmlProcessingInstruction`	Is a processing instruction node
Entity	`XmlEntity`	Represents the <!ENTITY...> declarations in an XML document, either from an internal document type definition (DTD) subset or from external DTDs and parameter entities
Notation	`XmlNotation`	Represents a notation declared in the DTD

XML Document Object Model

- The **`XmlNode`** class is the basic class in the DOM tree

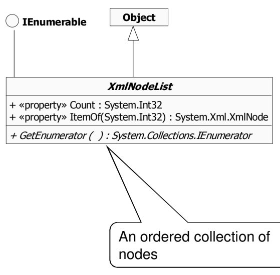


XML Document Object Model

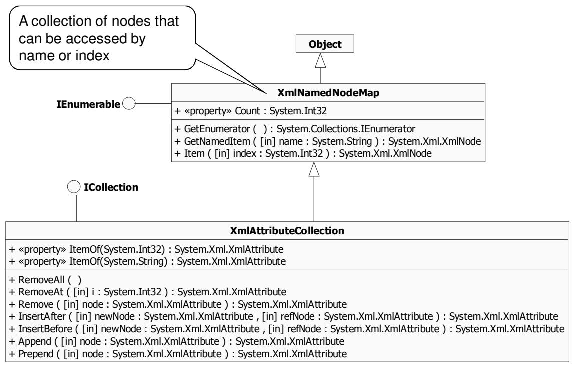

XmlNode
+ «property» Name : System.String
+ «property» Value : System.String
+ «property» NodeType : System.Xml.XmlNodeType
+ «property» ParentNode : System.Xml.XmlNode
+ «property» ChildNodes : System.Xml.XmlNodeList
+ «property» PreviousSibling : System.Xml.XmlNode
+ «property» NextSibling : System.Xml.XmlNode
+ «property» Attributes : System.Xml.XmlAttributeCollection
+ «property» OwnerDocument : System.Xml.XmlDocument
+ «property» FirstChild : System.Xml.XmlNode
+ «property» LastChild : System.Xml.XmlNode
+ «property» HasChildNodes : System.Boolean
+ «property» InnerText : System.String
+ «property» OuterXml : System.String
+ «property» InnerXml : System.String
+ RemoveAll ( )
+ Clone ( ) : System.Xml.XmlNode
+ CloneNode ([in] deep : System.Boolean) : System.Xml.XmlNode
+ AppendChild ([in] newChild : System.Xml.XmlNode) : System.Xml.XmlNode
+ PrependChild ([in] newChild : System.Xml.XmlNode) : System.Xml.XmlNode
+ RemoveChild ([in] oldChild : System.Xml.XmlNode) : System.Xml.XmlNode
+ ReplaceChild ([in] newChild : System.Xml.XmlNode , [in] oldChild : System.Xml.XmlNode) : System.Xml.XmlNode
+ InsertAfter ([in] newChild : System.Xml.XmlNode , [in] refChild : System.Xml.XmlNode) : System.Xml.XmlNode
+ InsertBefore ([in] newChild : System.Xml.XmlNode , [in] refChild : System.Xml.XmlNode) : System.Xml.XmlNode
+ SelectSingleNode ([in] xpath : System.String) : System.Xml.XmlNode
+ SelectNodes ([in] xpath : System.String) : System.Xml.XmlNodeList
+ GetEnumerator ( ) : System.Collections.IEnumerator
Object
IEnumerable
ICloneable
IXPathNavigable

```
virtual object ICloneable.Clone()
{ return CloneNode(true); }
public virtual XmlNode Clone()
{ return CloneNode(true); }
```

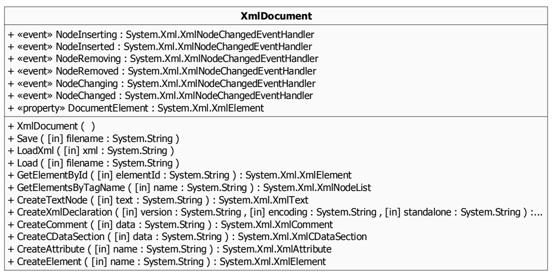
XML Document Object Model


IEnumerable
Object
XmlNodeList
+ «property» Count : System.Int32
+ «property» ItemOf(System.Int32) : System.Xml.XmlNode
+ GetEnumerator ( ) : System.Collections.IEnumerator
An ordered collection of nodes

XML Document Object Model


A collection of nodes that can be accessed by name or index
Object
IEnumerable
XmlNamedNodeMap
+ «property» Count : System.Int32
+ GetEnumerator ( ) : System.Collections.IEnumerator
+ GetNamedItem ([in] name : System.String) : System.Xml.XmlNode
+ Item ([in] index : System.Int32) : System.Xml.XmlNode
ICollection
XmlAttributeCollection
+ «property» ItemOf(System.Int32) : System.Xml.XmlAttribute
+ «property» ItemOf(System.String) : System.Xml.XmlAttribute
+ RemoveAll ( )
+ RemoveAt ([in] i : System.Int32) : System.Xml.XmlAttribute
+ Remove ([in] node : System.Xml.XmlAttribute) : System.Xml.XmlAttribute
+ InsertAfter ([in] newNode : System.Xml.XmlAttribute , [in] refNode : System.Xml.XmlAttribute) : System.Xml.XmlAttribute
+ InsertBefore ([in] newNode : System.Xml.XmlAttribute , [in] refNode : System.Xml.XmlAttribute) : System.Xml.XmlAttribute
+ Append ([in] node : System.Xml.XmlAttribute) : System.Xml.XmlAttribute
+ Prepend ([in] node : System.Xml.XmlAttribute) : System.Xml.XmlAttribute

XML Document Object Model


XmlDocument
+ «event» NodeInserting : System.Xml.XmlNodeChangedEventHandler
+ «event» NodeInserted : System.Xml.XmlNodeChangedEventHandler
+ «event» NodeRemoving : System.Xml.XmlNodeChangedEventHandler
+ «event» NodeRemoved : System.Xml.XmlNodeChangedEventHandler
+ «event» NodeChanging : System.Xml.XmlNodeChangedEventHandler
+ «event» NodeChanged : System.Xml.XmlNodeChangedEventHandler
+ «property» DocumentElement : System.Xml.XmlElement
+ XmlDocument ( )
+ Save ([in] filename : System.String)
+ LoadXml ([in] xml : System.String)
+ Load ([in] filename : System.String)
+ GetElementById ([in] elementId : System.String) : System.Xml.XmlElement
+ GetElementsByTagName ([in] name : System.String) : System.Xml.XmlNodeList
+ CreateTextNode ([in] text : System.String) : System.Xml.XmlText
+ CreateXmlDeclaration ([in] version : System.String , [in] encoding : System.String , [in] standalone : System.String) :...
+ CreateComment ([in] data : System.String) : System.Xml.XmlComment
+ CreateCDataSection ([in] data : System.String) : System.Xml.XmlCDataSection
+ CreateAttribute ([in] name : System.String) : System.Xml.XmlAttribute
+ CreateElement ([in] name : System.String) : System.Xml.XmlElement

XML Document Object Model

- **Lettura** (sincrona) di un documento XML da file (in caso di errore: `XmlException`)

```
XmlDocument document = new XmlDocument();
document.Load(fileName);
```

- **Reperimento** elemento radice di un documento XML

```
XmlElement root = document.DocumentElement;
```

- **Creazione di un nuovo documento** XML

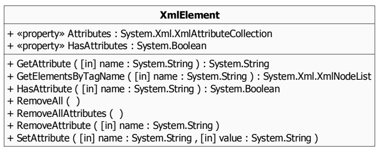
```
XmlDocument doc = new XmlDocument();
XmlNode node = doc.CreateXmlDeclaration("1.0","","");
doc.AppendChild(node);
XmlNode root = doc.CreateElement("XmlNodeType");
doc.AppendChild(root);
//  Inserimento di tutti gli altri nodi in root
```

- **Salvataggio** di un documento XML su file

```
doc.Save(fileName);
```

Esempio 3.1

XML Document Object Model



```
+ «property» Attributes : System.Xml.XmlAttributeCollection
+ «property» HasAttributes : System.Boolean
+ GetAttribute ( [in] name : System.String ) : System.String
+ GetElementsByTagName ( [in] name : System.String ) : System.Xml.XmlNodeList
+ HasAttribute ( [in] name : System.String ) : System.Boolean
+ RemoveAll (  )
+ RemoveAllAttributes (  )
+ RemoveAttribute ( [in] name : System.String )
+ SetAttribute ( [in] name : System.String , [in] value : System.String )
```

- **`GetAttribute(nomeAttributo)`**
  - Se l'attributo esiste, restituisce il valore dell'attributo
  - In caso contrario, restituisce una stringa vuota
- **`SetAttribute(nomeAttributo,valoreAttributo)`**
  - Se l'attributo esiste, ne cambia il valore
  - In caso contrario, crea un nuovo attributo con il valore specificato
- **`RemoveAttribute(nomeAttributo)`**
  - Se l'attributo esiste, lo elimina
  - In caso contrario, nop

Esempio 3.2

XML Document Object Model

- **`XmlNodeList SelectNodes(string xpath);`**
  Selects a list of nodes matching the XPath expression
- **`XmlNode SelectSingleNode(string xpath);`**
  Selects the first **`XmlNode`** that matches the XPath expression

```
<?xml version="1.0" encoding="utf-8" ?>
<Gruppo>
  <Item id="1">Pippo</Item>
  <Item id="2">Topolino</Item>
  <Item id="5">Paperino</Item>
  <Item id="7">Gastone</Item>
</Gruppo>
```

- Semplici espressioni XPath:
  - **`/Gruppo/Item`** → restituisce tutti gli **`Item`**
  - **`/Gruppo/Item[@id >= 5]`** → restituisce 2 **`Item`**
  - **`/Gruppo/Item[text() = 'Topolino']`** → restituisce 1 **`Item`**

Esempio 3.3